Neural Ordinary Differential Equations

Krishna Kumar

University of Texas at Austin

krishnak@utexas.edu

Overview

Outline

Learning Objectives

- Understand the connection between ResNets and continuous dynamics
- Master the Neural ODE framework and adjoint method
- Implement ODENets for image classification
- Apply continuous normalizing flows for generative modeling
- Build latent ODE models for irregular time series

Open Notebook

The ResNet Formula

A residual network transforms hidden states layer by layer:

$$h_{t+1} = h_t + f(h_t, \theta_t) \tag{1}$$

where $t \in \{0, 1, \ldots, T\}$ indexes the layers.

The Key Question

What happens as we add more layers ($T \to \infty$) and take smaller steps?

The Euler Connection

The ResNet update is the **Euler discretization** of an ODE:

$$\frac{dh(t)}{dt} = f(h(t), t, \theta) \tag{2}$$

Key Insight

A ResNet with infinitely many infinitesimal layers $\equiv$ solving an ODE

Instead of specifying discrete layers, we parameterize the **derivative** of the hidden state using a neural network.

ResNet vs Neural ODE

Residual Network

- Discrete transformations
- Fixed number of layers T
- $h_{t+1} = h_t + f(h_t)$
- Memory: $\mathcal{O}(T)$

Neural ODE

- Continuous dynamics
- Adaptive depth
- $\frac{dh}{dt} = f(h(t), t)$
- Memory: $\mathcal{O}(1)$

Outline

Three Key Advantages

1. **Memory Efficiency**: $\mathcal{O}(1)$ vs $\mathcal{O}(L)$
   - Adjoint method recomputes forward pass during backprop
   - Train arbitrarily deep networks with constant memory
2. **Adaptive Computation**
   - ODE solver adjusts # function evaluations automatically
   - Trade speed for accuracy via tolerance
3. **Continuous Time**
   - Natural for irregular time series
   - No discretization artifacts

Memory Efficiency: The Adjoint Method

Standard Backpropagation:

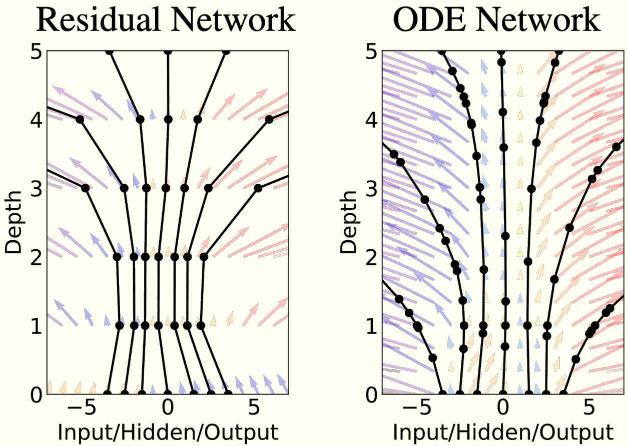
- Store all intermediate layer activations
- Memory: $\mathcal{O}(L)$ where $L =$ number of layers

Adjoint Method:

- Solve a second ODE backwards in time
- Recompute forward states during backward pass
- Memory: $\mathcal{O}(1)$ independent of depth

The adjoint state $a(t) = \frac{\partial L}{\partial h(t)}$ evolves as:

$$\frac{da(t)}{dt} = -a(t)^T \frac{\partial f(h(t), t, \theta)}{\partial h} \tag{3}$$



- **Forward**: Solve ODE from t_0 to t_1
- **Backward**: Solve augmented ODE from t_1 to t_0
- Automatically handled by `odeint_adjoint`

Outline

Basic Neural ODE Components

1. ODE Function $f(h, t, \theta)$

Neural network that computes the derivative $\frac{dh}{dt}$

2. ODE Solver

Integrates $h(t)$ from t_0 to t_1:

$$h(t_1) = h(t_0) + \int_{t_0}^{t_1} f(h(t), t, \theta) dt \tag{4}$$

3. Adjoint Method

Computes gradients $\frac{\partial L}{\partial \theta}$ efficiently

PyTorch Implementation

Using torchdiffeq

```
from torchdiffeq import odeint_adjoint as odeint

class ODEFunc(nn.Module):
    def forward(self, t, h):
        return self.net(h)

class NeuralODE(nn.Module):
    def forward(self, h0, t):
        return odeint(self.odefunc, h0, t)
```

Key parameters:

- `method`: 'dopri5' (adaptive), 'euler', 'rk4', etc.
- `rtol`, `atol`: Tolerance for accuracy

Hyperparameter Selection

ODE Solver Tolerance

- `rtol`, `atol`: Control accuracy
- Higher tolerance $\rightarrow$ faster but less accurate
- Typical: `rtol=1e-3`, `atol=1e-4`

Solver Method

- **Adaptive**: 'dopri5', 'adams' (recommended)
- **Fixed-step**: 'euler', 'rk4' (for debugging)

Integration Time

- Usually $T = 1.0$ (can be learned)
- Longer $T \rightarrow$ more expressive but slower

Extensions

Augmented Neural ODEs

Add extra dimensions to avoid topological constraints

Second-order Neural ODEs

Include acceleration: $\frac{d^2h}{dt^2} = f(h, \frac{dh}{dt}, t)$

Stochastic Differential Equations (SDEs)

Add noise for uncertainty: $dh = f(h,t)dt + g(h,t)dW$

Hamiltonian Neural Networks

Preserve energy and symplectic structure

Summary

Key Takeaways

1. Neural ODEs = continuous-depth neural networks
2. ResNets $\rightarrow$ ODEs via Euler discretization
3. Adjoint method enables $\mathcal{O}(1)$ memory training
4. Applications: classification, generative models, time series

The Big Idea

Parameterize the **derivative** of hidden states, not the states themselves

References

Chen, R. T. Q., Rubanova, Y., Bettencourt, J., & Duvenaud, D. (2018).
Neural Ordinary Differential Equations.
NeurIPS 2018 (Best Paper Award).

Grathwohl, W., Chen, R. T. Q., Bettencourt, J., Sutskever, I., & Duvenaud, D. (2019).
FFJORD: Free-form Continuous Dynamics for Scalable Reversible Generative Models.
ICLR 2019.

Rubanova, Y., Chen, R. T. Q., & Duvenaud, D. (2019).
Latent ODEs for Irregularly-Sampled Time Series.
NeurIPS 2019.